My Toy Package Documentation

Release 0.9.0

François Durand

Jan 22, 2020

Contents:

1 My Toy Package **1**
- 1.1 Credits . . . 1

2 Tutorial **3**
- 2.1 Preliminaries . . . 3
- 2.2 Create your package . . . 5
- 2.3 During the Life of Your Package . . . 9

3 Installation **11**
- 3.1 Stable release . . . 11
- 3.2 From sources . . . 11

4 Usage **13**

5 Reference **15**
- 5.1 A Nice Section . . . 15
- 5.2 Another Nice Section . . . 17

6 Contributing **19**
- 6.1 Types of Contributions . . . 19
- 6.2 Get Started! . . . 20
- 6.3 Pull Request Guidelines . . . 21
- 6.4 Tips . . . 21
- 6.5 Deploying . . . 21

7 Credits **23**
- 7.1 Development Lead . . . 23
- 7.2 Contributors . . . 23

8 History **25**
- 8.1 0.9.0 (2020-01-22) . . . 25
- 8.2 0.8.0 (2020-01-19) . . . 25
- 8.3 0.7.0 (2020-01-16) . . . 25
- 8.4 0.6.1 (2019-12-20) . . . 26
- 8.5 0.6.0 (2019-12-20) . . . 26
- 8.6 0.5.0 (2019-12-19) . . . 26
- 8.7 0.4.3 (2019-12-19) . . . 26

8.8 0.4.2 (2019-12-19) . . . 26
8.9 0.4.1 (2019-12-19) . . . 26
8.10 0.3.2 (2019-06-27) . . . 26
8.11 0.3.1 (2019-06-27) . . . 27
8.12 0.3.0 (2019-06-26) . . . 27
8.13 0.2.5 (2019-06-26) . . . 27
8.14 0.2.4 (2019-06-26) . . . 27
8.15 0.2.3 (2019-06-26) . . . 27
8.16 0.2.2 (2019-04-03) . . . 27
8.17 0.2.1 (2019-03-27) . . . 27
8.18 0.2.0 (2019-03-27) . . . 27
8.19 0.1.6 (2018-03-06) . . . 27
8.20 0.1.5 (2018-03-06) . . . 28
8.21 0.1.0 (2018-03-06) . . . 28

9 Indices and tables **29**

Index **31**

CHAPTER 1

My Toy Package

My Toy Package shows how to create and maintain a package.

- Free software: GNU General Public License v3.
- Documentation: https://my-toy-package.readthedocs.io.

The core of this package is a tutorial that gives a checklist of how to create and maintain your Python package, especially relying on Cookiecutter, by Audrey Roy Greenfeld, and PyCharm. We also use GitHub, ReadTheDocs, PyPI, Travis CI, Codecov and Pyup.

1.1 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Tutorial

In the end, here is how it will work.

- You use the PyCharm IDE. It is configured to run tests and generate the documentation of your package locally.
- Your project is on GitHub. When you push modifications to GitHub:
  - Travis CI automatically runs all the tests and checks that everything is working on several versions of Python (e.g. 3.5, 3.6, 3.7)
  - Codecov enables you to see what parts of your code are covered or not by your tests.
  - ReadTheDocs automatically generates the documentation and publishes it online.
- When you "tag" a version on GitHub, in other words when you "draft a release": Travis CI not only performs the tests but also generates the distribution files of your package and publishes them on PyPI. As a consequence, any Python user will be able to install you package via `pip install the_name_of_your_package`.
- Generally, you use some external packages during the development process of your own package, for example `sphinx`, `pytest`, etc. These packages and their versions are listed in the file `requirements_dev.txt` of your package, so that each member of your team know which version is used. PyUp informs you when a new version of these third-party packages are released: you receive a pull request in GitHub and you just have to accept it.

2.1 Preliminaries

2.1.1 Create accounts on the websites

Ensure that you have accounts (preferably with the same login) on:

- GitHub,
- ReadTheDocs,
- PyPI,
- Travis-CI,

- Codecov,
- Pyup.

2.1.2 Install Cookiecutter

In a terminal (e.g. Anaconda Prompt):

```
pip install cookiecutter
```

2.1.3 Install Git

If necessary, install git: https://git-scm.com/downloads . You may need to restart your computer.

2.1.4 Change the documentation style in PyCharm

Do this if you want to use Numpy style of documentation. In the "Welcome to PyCharm" window (before you open a project): Configure → Settings → Tools → Python Integrated Tools → Docstrings → Docstring format → NumPy.

2.1.5 Register your GitHub account in PyCharm

In PyCharm:

1. Menu File → Settings → Version Control → GitHub.
2. Click on the "+" icon.
3. Fill in the form and validate.

2.1.6 Register your GitHub account in ReadTheDocs

On ReadTheDocs website: Paramètres → Comptes liés. Check that your GitHub account is listed here.

2.1.7 Install Travis Client on your computer

- Under Windows:
  1. Install Ruby (https://rubyinstaller.org/).
  2. Run PyCharm as Administrator.
  3. In PyCharm terminal, do: `gem install -V travis`. If it does not work, restart your computer and try again.
- Under Debian, run as root:

```
apt-get update
apt-get install cookie-cutter ruby ruby-dev gcc
gem install -V travis
```

- Under Ubuntu 16, run:

```
sudo apt-get install ruby-dev
sudo gem install -V travis
```

If you experience troubles installing travis, cf. https://github.com/travis-ci/travis.rb#installation.

2.2 Create your package

This section is adapted from: https://cookiecutter-pypackage.readthedocs.io/en/latest/tutorial.html .

2.2.1 Generate Your Package

1. Your project will need a project name (e.g. *My Toy Package*) and a project slug (typically `my_toy_package`). Before starting, check that your project slug is not used in PyPI.
2. In a terminal (e.g. Anaconda Prompt):
    1. Go to the parent directory of where you want to put the directory of your package, e.g. `D:\GitHub\`.
    2. `cookiecutter https://github.com/francois-durand/cookiecutter-my_toy_package.git`
    3. Answer the questions. Here is an example (some explanations follow):

```
full_name [F. Durand]: François Durand
email [fradurand@gmail.com]: fradurand@gmail.com
github_username [francois-durand]: francois-durand
project_name [Python Boilerplate]: My Toy Package
project_slug [my_toy_package]:
project_short_description [Python Boilerplate contains all the boilerplate
you need to create a Python package.]: My Toy Package shows how to
create and maintain a package.
pypi_username [francois-durand]:
version [0.1.0]:
use_pytest [y]:
use_codecov [y]:
use_pypi_deployment_with_travis [y]:
add_pyup_badge [n]:
Select command_line_interface:
1 - No command-line interface
2 - Click
3 - Argparse
Choose from 1, 2, 3 (1, 2, 3) [1]:
create_author_file [y]:
Select open_source_license:
1 - GNU General Public License v3
2 - MIT license
3 - BSD license
4 - ISC license
5 - Apache Software License 2.0
6 - Not open source
Choose from 1, 2, 3, 4, 5, 6 (1, 2, 3, 4, 5, 6) [1]:
```

Some explanations now:

- `use_pytest`: there are essentially three ways to do unit tests in Python: unittest (the standard solution), pytest (the recommended test package) and doctest (where tests are integrated in the docstrings). If you are new to

testing, I suggest using doctest. But even so, pytest is useful to configure your tests (as we will do a bit later). For this reason, in all cases, my advice is to answer yes.

- `use_codecov`: you will use Codecov to assess the coverage of your code by your tests.
- `use_pypi_deployment_with_travis`: when you will do a *release* in GitHub, Travis will automatically release your package on PyPI.
- `add_pyup_badge`: a pyup badge will appear in the readme of your package. I suggest to answer no.
- `Click`: this allows you to easily call your program with unix-style command, e.g. `python my_program.py --help`. `Argparse` provides the same kind of feature. You can choose either of them, even if you do not use it for the moment. But personally, I answer no.
- `create_author_file`: I suggest to answer yes.

2.2.2 Create the PyCharm Project

In PyCharm:

1. Create new project.
2. In *Location*, fetch the directory of your project, e.g. `D:\GitHub\my_toy_package`. Validate.
3. Warning that the directory is not empty: validate.

2.2.3 Create the GitHub Repo

In PyCharm:

1. Menu VCS → Import into version control → Share project on GitHub.
2. Fill in the form and validate, e.g.:

```
New repository name: my_toy_package
Remote name: origin
Description: My Toy Package shows how to create and maintain a package.
```

In a browser, you can go to your GitHub account to check that everything is there.

N.B.: if you use a public GitHub repository, using PyPI is free (but not for a private repository).

2.2.4 Create a virtual environment

A virtual environment is essentially a Python installation dedicated to your project, with its own versions of the third-party packages. It ensures that if you reuse this project several months later, it will still work... This is not mandatory, but I suggest it especially if you use a third-party package that is still in a 0.x.x release (which means that its API is not considered stable yet).

1. Menu File → Settings → Project → Project Interpreter. (For Apple users: PyCharm → Preferences → Project → Project Interpreter.)
2. Click on the gear-shaped icon → Add.
3. Fill in the form: New environment using Virtualenv. This directory proposed is just fine. Validate.

2.2.5 Install Dev Requirements

In the PyCharm terminal:

1. Ensure you are in the directory of your package (e.g. `D:\GitHub\my_toy_package`).
2. If you have set a virtual environment, ensure that it is activated: there should be `(venv)` at the beginning of the line. If not:

```
Windows: venv\Scripts\activate
Linux:   source venv/bin/activate
```

3. `pip install -r requirements_dev.txt`

2.2.6 Install Your Package in "Development Mode"

This way, your package behaves as if it were installed, but any change you make will have effect immediately. In the PyCharm terminal, you should still be in the directory of your package, with your virtual environment activated. Do:

```
python setup.py develop
```

2.2.7 Add a Run Configuration for Doctest

In PyCharm:

1. Menu Run → Edit Configurations.
2. Add a new configuration by clicking the + button → Python tests → pytest.
3. Give a name to the configuration, e.g. `All tests`.
4. Ignore the warning and validate.

Run this configuration: normally, it runs all the tests of the project.

2.2.8 Add a Run Configuration for Sphinx

In PyCharm:

1. In the root of your project, add a directory named `build`.
2. Menu Run → Edit Configurations.
3. Plus icon (top left) → Python docs → Sphinx task.
4. Give a name to the configuration, e.g. `Generate docs`.
5. Input: the "docs" directory of your project.
6. Output: the "build" directory of your project.
7. OK.

Run this configuration: you should have a warning "Title underline too short". Go to the mentioned file and correct the problem. Then run the configuration again: normally, it generates the documentation. To check the result, you can open the file `build/index.html`.

2.2.9 Set Up ReadTheDocs

1. On ReadTheDocs website:
    1. Go to "My Projects". Import a Project → Importer manuellement. Fill in the form and validate, e.g.:

    ```
    my_toy_package
    https://github.com/francois-durand/my_toy_package
    Git
    ```

    2. Admin → Advanced settings. Check "Installer votre projet dans un virtualenv via setup.py install".

2.2.10 Set Up Pyup

If you work on a "small" project, I suggest that you do not use pyup: it will just generate a lot of spam in your email inbox. However, for a more ambitious project, it may be useful.

1. On Pyup website:
    1. Click on the green *Add Repo* button and select the repo you created.
    2. A pop up appears. Personally, I checked the first item and unchecked the two others.

    Within a few minutes, you will probably receive a pull request in GitHub (and in your email).
2. On GitHub website, open the pull request and:
    1. Merge pull request.
    2. Accept merge.
    3. Delete branch.
3. In PyCharm, menu VCS → Update project. This does a git update (to get the modifications done by Pyup).

2.2.11 Set Up Travis CI

1. On Travis website:
    1. Login using your Github credentials.
    2. It may take a few minutes for Travis CI to load up a list of all your GitHub repos. If you do not see your new repo, log out and log in again.
    3. Click on your new repo.
    4. Click on "Activate repository".
2. In PyCharm terminal, ensure that you are in the directory of your project and:

    ```
    travis encrypt --add deploy.password "My PyPI password"
    ```

    (replace with your actual password, in quotation marks).
3. Open the file `.travis.yml`, which is in the root of your project (you can do so in PyCharm). Check that `deploy.password.secure` is encoded.

2.2.12 Check that Everything is Working

1. In PyCharm: commit/push if necessary, i.e.:
    1. Menu VCS → Commit.
    2. Enter a commit message.
    3. Commit → Commit and push.
    4. Push.
2. In Travis CI website: go to Build History. The build should be a success (it may take several minutes).
3. In Codecov website: once Travis has finished building, you can navigate in your project to see what parts of the code are covered by the tests.
4. In ReadTheDocs website:
    1. In *Compilations*, the doc should be *transmis*.
    2. Open the documentation.
    3. In the table of contents, click on the first page (e.g. *My Toy Package*). Depending on your initial choice of options, you should have three to five *badges*:
        1. PyPI: invalid (there will be the version number after your first release).
        2. Build: passing.
        3. Docs: passing.
        4. Codecov (optional): with a percentage.
        5. Pyup (optional): up-to-date.
    4. In the table of contents, click on *Reference*. You should see the doc of your functions.

If you wish, you are now ready to release your first version (cf. below).

2.3 During the Life of Your Package

2.3.1 Release a Version

In PyCharm:

1. Run the tests.
2. Generate the documentation locally in order to check that it is working.
3. Update the file `HISTORY.rst`.
4. Check that the readme will be correctly rendered on PyPI. In a terminal:

```
python setup.py bdist
twine check dist/the_name_of_the_file.zip
```

   where `the_name_of_the_file` must be replaced by the relevant file name.
5. Commit/push.
6. In PyCharm terminal, do one of the following:

- `bumpversion patch` (version x.y.z → x.y.(z+1)) when you made a backwards-compatible modification (such as a bug fix).
- `bumpversion minor` (version x.y.z → x.(y+1).0) when you added a functionality.
- `bumpversion major` (version x.y.z → (x+1).0.0) when you changed the API. Note: in versions 0.y.z, the API is not expected to be stable anyway.

If you were working on a secondary branch, do what you have to (pull request to master, etc).

On Github website, go to "releases". Select "Draft a new release", add a tag name (e.g. `v0.1.0`) and a message (e.g. `First stable version`). Select "Publish release".

After a few minutes, Travis CI has finished the built and it is deployed on PyPI.

2.3.2 Add a Module (= a File)

Typically, this is a file `SubPackage\MyClass`, containing class `MyClass`.

1. In the file `__init__.py`: add the shortcut.
2. In the file `reference.rst`: add the auto-documentation.

2.3.3 Use a Third-Party Package

For example, you want to use Numpy in your module.

In the file `setup.py`, in the list `requirements`, add the name of the package (e.g. `'numpy`).

2.3.4 When You Receive a Pull Request from Pyup

1. In GitHub website:
    1. Open the pull request.
    2. If necessary, wait until Travis CI has finished the build, so that you know there is no problem.
    3. Merge pull request.
    4. Confirm merge.
    5. Delete branch.
    6. In the front page, you Pyup badge should be up-to-date. If not, this is probably just a matter of time. You can go to the Pyup website, click on the gear → reload.
2. In PyCharm, Menu VCS → Update project.

CHAPTER 3

Installation

3.1 Stable release

To install My Toy Package, run this command in your terminal:

```
$ pip install my_toy_package
```

This is the preferred method to install My Toy Package, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

3.2 From sources

The sources for My Toy Package can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/francois-durand/my_toy_package
```

Or download the tarball:

```
$ curl  -OL https://github.com/francois-durand/my_toy_package/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 4

Usage

To use My Toy Package in a project:

```
import my_toy_package
```

CHAPTER 5

Reference

5.1 A Nice Section

class my_toy_package.**MyClass1**(*a: float*, *b: float*)

A whatever-you-are-doing.

Parameters

- **a** (*float*) – The *a* of the system. Must be non-negative.
- **b** (*float*) – The *b* of the system.

my_string

A nice string.

Type str

Raises ValueError – If *a* is negative.

Notes

Document the `__init__()` method in the docstring of the class itself, because the docstring of the `__init__()` method does not appear in the documentation.

- Refer to a class this way: *MyClass2* (except as a type indication, cf. *update_b_from_class_2()*).
- Refer to a method this way: *addition()*.
- Refer to a method in another class: *MyClass2.addition()*.
- Refer to an attribute this way: *my_string*.
- Refer to a property this way: *a_square*.
- Refer to a parameter or variable this way: *a*.

Examples

```
>>> my_object = MyClass1(a=5, b=3)
```

A_NICE_CONSTANT = 42
This is a nice constant.

A_VERY_NICE_CONSTANT = 51

a_square
The square of *a*.

addition() → float
Add *a* and *b*.

Returns The sum of *a* and *b*.

Return type Number

Examples

```
>>> my_object = MyClass1(a=5, b=3)
>>> my_object.addition()
8
```

divide_a_by_c_and_add_d(*c: float*, *d: float*) → float
Divide *a* by something and add something else.

Parameters

- **c** (*Number*) – A non-zero number. You can say many things about this parameter in several indented lines, like this.
- **d** (*Number*) – A beautiful number.

Returns The result of *a* / *c* + *d*.

Return type Number

Raises ZeroDivisionError – If *c* = 0.

Notes

This function gives an example of documentation with typical features.

Examples

We can write some text to explain the following example:

```
>>> my_object = MyClass1(a=5, b=3)
>>> my_object.divide_a_by_c_and_add_d(c=2, d=10)
12.5
```

And we can explain a second example here:

```
>>> my_object = MyClass1(a=5, b=3)
>>> my_object.divide_a_by_c_and_add_d(c=2, d=20)
22.5
```

update_b_from_class_2(*object_of_class_2*)
Update *b* from a *MyClass2* object.

Parameters **object_of_class_2** (MyClass2) – An object from the other class. The purpose of this function is essentially to show how to document when an argument is an object of another class.

N.B.: for the type of an argument, you can enter only the name of the class, e.g. `MyClass2`. However, in the rest of the documentation, you must use the full syntax, like `:class:`MyClass2``.

Examples

```
>>> my_object = MyClass1(a=5, b=3)
>>> my_object.update_b_from_class_2(MyClass2(42, 51))
>>> my_object.b
51
```

5.2 Another Nice Section

class my_toy_package.**MyClass2**(*a: float*, *b: float*)
A whatever-you-are-doing.

Parameters

- **a** (*float*) – The *a* of the system.
- **b** (*float*) – The *b* of the system.

Examples

```
>>> my_object = MyClass2(a = 5, b = 3)
```

addition() → float
Add *a* and *b*.

Returns The sum of *a* and *b*.

Return type Float

Examples

```
>>> my_object = MyClass2(a=5, b=3)
>>> my_object.addition()
8
```

class my_toy_package.**MyClass3**(*a: float*, *b: float*)
A whatever-you-are-doing.

Parameters

- **a** (*float*) – The *a* of the system.
- **b** (*float*) – The *b* of the system.

Examples

```
>>> my_object = MyClass3(a = 5, b = 3)
```

addition() → float
Add *a* and *b*.

Returns The sum of *a* and *b*.

Return type Float

Examples

```
>>> my_object = MyClass3(a=5, b=3)
>>> my_object.addition()
8
```

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/francois-durand/my_toy_package/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

6.1.4 Write Documentation

My Toy Package could always use more documentation, whether as part of the official My Toy Package docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/francois-durand/my_toy_package/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *my_toy_package* for local development.

1. Fork the *my_toy_package* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/my_toy_package.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv my_toy_package
$ cd my_toy_package/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 my_toy_package tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/francois-durand/my_toy_package/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ py.test tests.test_my_toy_package
```

6.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 7

Credits

7.1 Development Lead

- François Durand <fradurand@gmail.com>

7.2 Contributors

None yet. Why not be the first?

CHAPTER 8

History

8.1 0.9.0 (2020-01-22)

- The cookiecutter now features a pytest configuration in `tox.ini` and a file `.coveragerc`.
- As a consequence, it is not necessary anymore to add `--doctest-modules` when configuring the tests in PyCharm.

8.2 0.8.0 (2020-01-19)

- Create our own cookiecutter: `francois-durand/cookiecutter-my_toy_package`.
- In the tutorial:
  - Remove the steps that are now unnecessary, because they are included in the cookiecutter.
  - Create a section "Preliminaries" for the steps that are needed only once, not for every package creation.
  - Reorganize the order of the steps into a more natural course of action.
  - In PyCharm, change the documentation style to NumPy for all future package creations.

8.3 0.7.0 (2020-01-16)

- Configure Codecov and add a Codecov badge.
- Reach 100% of code coverage.
- In the tutorial, explain how to configure Codecov.

8.4 0.6.1 (2019-12-20)

- Add `long_description_content_type` in `setup.py` to avoid a warning in PyPI.

8.5 0.6.0 (2019-12-20)

- You may need to restart your computer after installing git.
- Cookiecutter now proposes argparse in addition to Click.
- It is not necessary to add twine to `requirements_dev.txt` (cookiecutter now does it).
- Update the procedure to install travis.
- It is not necessary anymore to remove mentions of Python 2.7 (cookiecutter has removed them).
- Remove the line `modules` from `reference.rst`.
- Add ReadTheDocs theme in `conf.py`.
- Create the directory `build` before setting up sphinx locally.

8.6 0.5.0 (2019-12-19)

- Explain how to indicate the type of an argument in the docstring when it is an object of one of your classes.

8.7 0.4.3 (2019-12-19)

- Correct the format of titles in `HISTORY.rst` to comply with PyPI's demands.

8.8 0.4.2 (2019-12-19)

- Separate the tutorial from the readme file, in the hope that it will solve the deployment problem on PyPI.

8.9 0.4.1 (2019-12-19)

- Use numpy style of documentation instead of sphinx basic style.
- In the readme, correct the explanations about Pyup.
- Say more explicitly that some steps are optional, like setting a virtual environment or using pyup.
- Added how to make travis run the doctests (thanks to Quentin Lutz).
- Remove the version numbers from the dev requirements.

8.10 0.3.2 (2019-06-27)

- Try to deploy again on PyPI.

8.11 0.3.1 (2019-06-27)

- Try to deploy again on PyPI.

8.12 0.3.0 (2019-06-26)

- Try to change the minor version number to solve deployment problem on PyPI.

8.13 0.2.5 (2019-06-26)

- Downgrade dev requirements to try to solve deployment problem on PyPI.

8.14 0.2.4 (2019-06-26)

- Try to tackle deployment problem on PyPI.

8.15 0.2.3 (2019-06-26)

- Correct the procedure for version release.

8.16 0.2.2 (2019-04-03)

- Minor updates in documentation.

8.17 0.2.1 (2019-03-27)

- Update flake.

8.18 0.2.0 (2019-03-27)

- Configuration for local build of documentation with Sphinx.
- Release a version directly on Github's website.
- Minor edits in documentation.

8.19 0.1.6 (2018-03-06)

- Minor edit in documentation.

8.20 0.1.5 (2018-03-06)

- Patch upload subpackages.

8.21 0.1.0 (2018-03-06)

- First release on PyPI.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Index

A

A_NICE_CONSTANT (*my_toy_package.MyClass1 attribute*), 16
a_square (*my_toy_package.MyClass1 attribute*), 16
A_VERY_NICE_CONSTANT (*my_toy_package.MyClass1 attribute*), 16
addition() (*my_toy_package.MyClass1 method*), 16
addition() (*my_toy_package.MyClass2 method*), 17
addition() (*my_toy_package.MyClass3 method*), 18

D

divide_a_by_c_and_add_d() (*my_toy_package.MyClass1 method*), 16

M

my_string (*my_toy_package.MyClass1 attribute*), 15
MyClass1 (*class in my_toy_package*), 15
MyClass2 (*class in my_toy_package*), 17
MyClass3 (*class in my_toy_package*), 17

U

update_b_from_class_2() (*my_toy_package.MyClass1 method*), 16